PROGRAMMABLE CIRCUIT AND RELATED COMPUTING MACHINE AND METHOD

CLAIM OF PRIORITY

[1] This application claims priority to U.S. Provisional Application Serial No. 60/422,503, filed on October 31, 2002, which is incorporated by reference.

CROSS REFERENCE TO RELATED APPLICATIONS

[2] This application is related to U.S. Patent App. Ser. Nos. ____ entitled IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-11-3), ______ entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-12-3); ______ entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-13-3) and __ entitled PIPELINE ACCELERATOR HAVING MULTIPLE PIPELINE UNITS AND RELATED COMPUTING MACHINE AND METHOD (Attorney Docket No. 1934-15-3), which have a common filing date and owner and which are incorporated by reference.

BACKGROUND

[3] A common computing architecture for processing relatively large amounts of data in a relatively short period of time includes multiple interconnected processors that share the processing burden. By sharing the processing burden, these multiple processors can often process the data more quickly than a single processor can for a given clock frequency. For example, each of the processors can process a respective portion of the data or execute a respective portion of a processing algorithm.

[4] **FIG. 1** is a schematic block diagram of a conventional computing machine ***10*** having a multi-processor architecture. The machine ***10*** includes a master processor ***12*** and coprocessors $\mathbf{14_1} - \mathbf{14_n}$, which communicate with each other and the master processor via a bus ***16***, an input port ***18*** for receiving raw data from a remote device (not shown in **FIG. 1**), and an output port ***20*** for providing processed data to the remote source. The machine ***10*** also includes a memory ***22*** for the master

processor ***12***, respective memories ***24***$_1$ – ***24***$_n$ for the coprocessors ***14***$_1$ – ***14***$_n$, and a memory ***26*** that the master processor and coprocessors share via the bus ***16***. The memory ***22*** serves as both a program and a working memory for the master processor ***12***, and each memory ***24***$_1$ – ***24***$_n$ serves as both a program and a working memory for a respective coprocessor ***14***$_1$ – ***14***$_n$. The shared memory ***26*** allows the master processor ***12*** and the coprocessors ***14*** to transfer data among themselves, and from/to the remote device via the ports ***18*** and ***20***, respectively. The master processor ***12*** and the coprocessors ***14*** also receive a common clock signal that controls the speed at which the machine ***10*** processes the raw data.

[5] In general, the computing machine ***10*** effectively divides the processing of raw data among the master processor ***12*** and the coprocessors ***14***. The remote source (not shown in **FIG. 1**) such as a sonar array loads the raw data via the port ***18*** into a section of the shared memory ***26***, which acts as a first-in-first-out (FIFO) buffer (not shown) for the raw data. The master processor ***12*** retrieves the raw data from the memory ***26*** via the bus ***16***, and then the master processor and the coprocessors ***14*** process the raw data, transferring data among themselves as necessary via the bus ***16***. The master processor ***12*** loads the processed data into another FIFO buffer (not shown) defined in the shared memory ***26***, and the remote source retrieves the processed data from this FIFO via the port ***20***.

[6] In an example of operation, the computing machine ***10*** processes the raw data by sequentially performing n + 1 respective operations on the raw data, where these operations together compose a processing algorithm such as a Fast Fourier Transform (FFT). More specifically, the machine ***10*** forms a data-processing pipeline from the master processor ***12*** and the coprocessors ***14***. For a given frequency of the clock signal, such a pipeline often allows the machine ***10*** to process the raw data faster than a machine having only a single processor.

[7] After retrieving the raw data from the raw-data FIFO (not shown) in the memory ***26***, the master processor ***12*** performs a first operation, such as a trigonometric function, on the raw data. This operation yields a first result, which the processor ***12***

stores in a first-result FIFO (not shown) defined within the memory ***26***. Typically, the processor ***12*** executes a program stored in the memory ***22***, and performs the above-described actions under the control of the program. The processor ***12*** may also use the memory ***22*** as working memory to temporarily store data that the processor generates at intermediate intervals of the first operation.

[8] Next, after retrieving the first result from the first-result FIFO (not shown) in the memory ***26***, the coprocessor $\boldsymbol{14_1}$ performs a second operation, such as a logarithmic function, on the first result. This second operation yields a second result, which the coprocessor $\boldsymbol{14_1}$ stores in a second-result FIFO (not shown) defined within the memory ***26***. Typically, the coprocessor $\boldsymbol{14_1}$ executes a program stored in the memory $\boldsymbol{24_1}$, and performs the above-described actions under the control of the program. The coprocessor $\boldsymbol{14_1}$ may also use the memory $\boldsymbol{24_1}$ as working memory to temporarily store data that the coprocessor generates at intermediate intervals of the second operation.

[9] Then, the coprocessors $\boldsymbol{24_2} - \boldsymbol{24_n}$ sequentially perform third – n^{th} operations on the second – $(n-1)^{th}$ results in a manner similar to that discussed above for the coprocessor $\boldsymbol{24_1}$.

[10] The n^{th} operation, which is performed by the coprocessor $\boldsymbol{24_n}$, yields the final result, *i.e.,* the processed data. The coprocessor $\boldsymbol{24_n}$ loads the processed data into a processed-data FIFO (not shown) defined within the memory ***26***, and the remote device (not shown in **FIG. 1**) retrieves the processed data from this FIFO.

[11] Because the master processor ***12*** and coprocessors ***14*** are simultaneously performing different operations of the processing algorithm, the computing machine ***10*** is often able to process the raw data faster than a computing machine having a single processor that sequentially performs the different operations. Specifically, the single processor cannot retrieve a new set of the raw data until it performs all n + 1 operations on the previous set of raw data. But using the pipeline technique discussed above, the master processor ***12*** can retrieve a new set of raw data after performing only the first operation. Consequently, for a given clock frequency, this

pipeline technique can increase the speed at which the machine ***10*** processes the raw data by a factor of approximately n + 1 as compared to a single-processor machine (not shown in **FIG. 1**).

[12] Alternatively, the computing machine ***10*** may process the raw data in parallel by simultaneously performing n + 1 instances of a processing algorithm, such as an FFT, on the raw data. That is, if the algorithm includes n + 1 sequential operations as described above in the previous example, then each of the master processor ***12*** and the coprocessors ***14*** sequentially perform all n + 1 operations on respective sets of the raw data. Consequently, for a given clock frequency, this parallel-processing technique, like the above-described pipeline technique, can increase the speed at which the machine ***10*** processes the raw data by a factor of approximately n + 1 as compared to a single-processor machine (not shown in **FIG. 1**).

[13] Unfortunately, although the computing machine ***10*** can process data more quickly than a single-processor computing machine (not shown in **FIG. 1**), the data-processing speed of the machine ***10*** is often significantly less than the frequency of the processor clock. Specifically, the data-processing speed of the computing machine ***10*** is limited by the time that the master processor ***12*** and coprocessors ***14*** require to process data. For brevity, an example of this speed limitation is discussed in conjunction with the master processor ***12***, although it is understood that this discussion also applies to the coprocessors ***14***. As discussed above, the master processor ***12*** executes a program that controls the processor to manipulate data in a desired manner. This program includes a sequence of instructions that the processor ***12*** executes. Unfortunately, the processor ***12*** typically requires multiple clock cycles to execute a single instruction, and often must execute multiple instructions to process a single value of data. For example, suppose that the processor ***12*** is to multiply a first data value A (not shown) by a second data value B (not shown). During a first clock cycle, the processor ***12*** retrieves a multiply instruction from the memory ***22***. During second and third clock cycles, the processor ***12*** respectively retrieves A and B from the memory ***26***. During a fourth clock cycle, the processor ***12*** multiplies A and B, and, during a fifth clock cycle, stores the resulting product in the memory ***22*** or ***26*** or provides the resulting

product to the remote device (not shown). This is a best-case scenario, because in many cases the processor ***12*** requires additional clock cycles for overhead tasks such as initializing and closing counters. Therefore, at best the processor ***12*** requires five clock cycles, or an average of 2.5 clock cycles per data value, to process A and B..

[14] Consequently, the speed at which the computing machine ***10*** processes data is often significantly lower than the frequency of the clock that drives the master processor ***12*** and the coprocessors ***14***. For example, if the processor ***12*** is clocked at 1.0 Gigahertz (GHz) but requires an average of 2.5 clock cycles per data value, then the effective data-processing speed equals (1.0 GHz)/2.5 = 0.4 GHz. This effective data-processing speed is often characterized in units of operations per second. Therefore, in this example, for a clock speed of 1.0 GHz, the processor ***12*** would be rated with a data-processing speed of 0.4 Gigaoperations/second (Gops).

[15] **FIG. 2** is a block diagram of a hardwired data pipeline ***30*** that can typically process data faster than a processor can for a given clock frequency, and often at substantially the same rate at which the pipeline is clocked. The pipeline ***30*** includes operator circuits $\boldsymbol{32_1} - \boldsymbol{32_n}$, which each perform a respective operation on respective data without executing program instructions. That is, the desired operation is "burned in" to a circuit ***32*** such that it implements the operation automatically, without the need of program instructions. By eliminating the overhead associated with executing program instructions, the pipeline ***30*** can typically perform more operations per second than a processor can for a given clock frequency.

[16] For example, the pipeline ***30*** can often solve the following equation faster than a processor can for a given clock frequency:

$$Y(x_k) = (5x_k + 3)2^{xk}$$

where x_k represents a sequence of raw data values. In this example, the operator circuit $\boldsymbol{32_1}$ is a multiplier that calculates $5x_k$, the circuit $\boldsymbol{32_2}$ is an adder that calculates $5x_k + 3$, and the circuit $\boldsymbol{32_n}$ (n = 3) is a multiplier that calculates $(5x_k + 3)2^{xk}$.

[17] During a first clock cycle k=1, the circuit $\boldsymbol{32_1}$ receives data value x_1 and multiplies it by 5 to generate $5x_1$.

[18] During a second clock cycle k = 2, the circuit ***32_2*** receives $5x_1$ from the circuit ***32_1*** and adds 3 to generate $5x_1 + 3$. Also, during the second clock cycle, the circuit ***32_1*** generates $5x_2$.

[19] During a third clock cycle k = 3, the circuit ***32_3*** receives $5x_1 + 3$ from the circuit ***32_2*** and multiplies by 2^{x1}(effectively left shifts $5x_1 + 3$ by x_1) to generate the first result $(5x_1 + 3)2^{x1}$. Also during the third clock cycle, the circuit ***32_1*** generates $5x_3$ and the circuit ***32_2*** generates $5x_2 + 3$.

[20] The pipeline ***30*** continues processing subsequent raw data values x_k in this manner until all the raw data values are processed.

[21] Consequently, a delay of two clock cycles after receiving a raw data value x_1 — this delay is often called the latency of the pipeline ***30*** — the pipeline generates the result $(5x_1 + 3)2^{x1}$, and thereafter generates one result — *e.g.*, $(5x_2 + 3)2^{x2}$, $(5x_3 + 3)2^{x3}$, . . ., $5x_n + 3)2^{xn}$ — each clock cycle.

[22] Disregarding the latency, the pipeline ***30*** thus has a data-processing speed equal to the clock speed. In comparison, assuming that the master processor ***12*** and coprocessors ***14*** (**FIG. 1**) have data-processing speeds that are 0.4 times the clock speed as in the above example, the pipeline ***30*** can process data 2.5 times faster than the computing machine ***10*** (**FIG. 1**) for a given clock speed.

[23] Still referring to **FIG. 2**, a designer may choose to implement the pipeline ***30*** in a programmable logic IC (PLIC), such as a field-programmable gate array (FPGA), because a PLIC allows more design and modification flexibility than does an application specific IC (ASIC). To configure the hardwired connections within a PLIC, the designer merely sets interconnection-configuration registers disposed within the PLIC to predetermined binary states. The combination of all these binary states is often called "firmware." Typically, the designer loads this firmware into a nonvolatile memory (not shown in **FIG. 2**) that is coupled to the PLIC. When one "turns on" the PLIC, it downloads the firmware from the memory into the interconnection-configuration registers. Therefore, to modify the functioning of the PLIC, the designer merely modifies the firmware and allows the PLIC to download the modified firmware into the

interconnection-configuration registers. This ability to modify the PLIC by merely modifying the firmware is particularly useful during the prototyping stage and for upgrading the pipeline ***30*** "in the field".

[24] Unfortunately, the hardwired pipeline ***30*** may not be the best choice to execute algorithms that entail significant decision making, particularly nested decision making. A processor can typically execute a nested-decision-making instruction (*e.g.,* a nested conditional instruction such as "if A, then do B, else if C, do D, . . ., else do n") approximately as fast as it can execute an operational instruction (*e.g.,* "A + B") of comparable length. But although the pipeline ***30*** may be able to make a relatively simple decision (*e.g.,* "A > B?") efficiently, it typically cannot execute a nested decision (*e.g.*, "if A, then do B, else if C, do D, . . ., else do n") as efficiently as a processor can. One reason for this inefficiency is that the pipeline ***30*** may have little on-board memory, and thus may need to access external working/program memory (not shown). And although one may be able to design the pipeline ***30*** to execute such a nested decision, the size and complexity of the required circuitry often makes such a design impractical, particularly where an algorithm includes multiple different nested decisions.

[25] Consequently, processors are typically used in applications that require significant decision making, and hardwired pipelines are typically limited to "number crunching" applications that entail little or no decision making.

[26] Furthermore, as discussed below, it is typically much easier for one to design/modify a processor-based computing machine, such as the computing machine ***10*** of **FIG. 1**, than it is to design/modify a hardwired pipeline such as the pipeline ***30*** of **FIG. 2**, particularly where the pipeline ***30*** includes multiple PLICs.

[27] Computing components, such as processors and their peripherals (*e.g.,* memory), typically include industry-standard communication interfaces that facilitate the interconnection of the components to form a processor-based computing machine.

[28] Typically, a standard communication interface includes two layers: a physical layer and a services layer.

[29] The physical layer includes the circuitry and the corresponding circuit interconnections that form the communication interface, and the operating parameters of this circuitry. For example, the physical layer includes the pins that connect the component to a bus, the buffers that latch data received from the pins, the drivers that drive signals onto the pins, and circuitry for recovering data from an input data signal and for recovering a clock signal from the data signal or from an external clock signal. The operating parameters include the acceptable voltage range of the data signals that the pins receive, the signal timing for writing and reading data, and the supported modes of operation (*e.g.*, burst mode, page mode). Conventional physical layers include transistor-transistor logic (TTL) and RAMBUS.

[30] The services layer includes the protocol by which a computing component transfers data. The protocol defines the format of the data and the manner in which the component sends and receives the formatted data. Conventional communication protocols include file-transfer protocol (FTP) and transmission control protocol/internet protocol (TCP/IP).

[31] Consequently, because manufacturers and others typically design computing components having industry-standard communication interfaces, one can typically design the interface of such a component and interconnect it to other computing components with relatively little effort. This allows one to devote most of his time to designing the other portions of the computing machine, and to easily modify the machine by adding or removing components.

[32] Designing a computing component that supports an industry-standard communication interface allows one to save design time by using an existing physical-layer design from a design library. This also insures that he/she can easily interface the component to off-the-shelf computing components.

[33] And designing a computing machine using computing components that support a common industry-standard communication interface allows the designer to interconnect the components with little time and effort. Because the components support a common interface, the designer can interconnect them via a system bus with

little design effort. And because the supported interface is an industry standard, one can easily modify the machine. For example, one can add different components and peripherals to the machine as the system design evolves, or can easily add/design next-generation components as the technology evolves. Furthermore, because the components support a common industry-standard services layer, one can incorporate into the computing machine's software an existing software module that implements the corresponding protocol. Therefore, one can interface the components with little effort because the interface design is essentially already in place, and thus can focus on designing the portions (*e.g.*, software) of the machine that cause the machine to perform the desired function(s).

[34] But unfortunately, there are no known industry-standard services layers for components, such as PLICs, used to form hardwired pipelines such as the pipeline ***30*** of **FIG. 2**.

[35] Consequently, to design a pipeline having multiple PLICs, one typically spends a significant amount of time and exerts a significant effort designing "from scratch" and debugging the services layer of the communication interface between the PLICs. Typically, such an ad hoc services layer depends on the parameters of the data being transferred between the PLICs. Likewise, to design a pipeline that interfaces to a processor, one would have to spend a significant amount of time and exert a significant effort in designing and debugging the services layer of the communication interface between the pipeline and the processor.

[36] Similarly, to modify such a pipeline by adding a PLIC to it, one typically spends a significant amount of time and exerts a significant effort designing and debugging the services layer of the communication interface between the added PLIC and the existing PLICs. Likewise, to modify a pipeline by adding a processor, or to modify a computing machine by adding a pipeline, one would have to spend a significant amount of time and exert a significant effort in designing and debugging the services layer of the communication interface between the pipeline and processor.

[37] Consequently, referring to **FIGS. 1** and **2**, because of the difficulties in interfacing multiple PLICs and in interfacing a processor to a pipeline, one is often forced to make significant tradeoffs when designing a computing machine. For example, with a processor-based computing machine, one is forced to trade number-crunching speed and design/modification flexibility for complex decision-making ability. Conversely, with a hardwired pipeline-based computing machine, one is forced to trade complex-decision-making ability and design/modification flexibility for number-crunching speed. Furthermore, because of the difficulties in interfacing multiple PLICs, it is often impractical for one to design a pipeline-based machine having more than a few PLICs. As a result, a practical pipeline-based machine often has limited functionality. And because of the difficulties in interfacing a processor to a PLIC, it would be impractical to interface a processor to more than one PLIC. As a result, the benefits obtained by combining a processor and a pipeline would be minimal.

[38] Therefore, a need has arisen for a new computing architecture that allows one to combine the decision-making ability of a processor-based machine with the number-crunching speed of a hardwired-pipeline-based machine.

SUMMARY

[39] According to an embodiment of the invention, a programmable circuit receives firmware from an external source, stores the firmware in a memory, and then downloads the firmware from the memory.

[40] Such a programmable circuit allows a system, such as a computing machine, to modify a programmable circuit's configuration, thus eliminating the need for manually reprogramming the configuration memory. For example, if the programmable circuit is an FPGA that is part of a pipeline accelerator, a processor coupled to the accelerator can modify the configuration of the FPGA. More specifically, the processor retrieves from a configuration registry firmware that represents the modified configuration, and sends the firmware to the FPGA, which then stores the firmware in a memory such as an electrically erasable and programmable read-only memory (EEPROM). Next, the FPGA downloads the firmware from the memory into its

configuration registers, and thus effectively reconfigures itself to have the modified configuration.

BRIEF DESCRIPTION OF THE DRAWINGS

[41] **FIG. 1** is a block diagram of a computing machine having a conventional multi-processor architecture.

[42] **FIG. 2** is a block diagram of a conventional hardwired pipeline.

[43] **FIG. 3** is a block diagram of a computing machine having a peer-vector architecture according to an embodiment of the invention.

[44] **FIG. 4** is a block diagram of a pipeline unit of the pipeline accelerator of **FIG. 3** according to an embodiment of the invention.

[45] **FIG. 5** is a diagram of a logical partitioning of the firmware memory of **FIG. 4** according to an embodiment of the invention.

[46] **FIG. 6** is a block diagram of a pipeline unit of the pipeline accelerator of **FIG. 3** according to another embodiment of the invention.

DETAILED DESCRIPTION

[47] **FIG. 3** is a schematic block diagram of a computing machine ***40***, which has a peer-vector architecture according to an embodiment of the invention. In addition to a host processor ***42***, the peer-vector machine ***40*** includes a pipeline accelerator ***44***, which performs at least a portion of the data processing, and which thus effectively replaces the bank of coprocessors ***14*** in the computing machine ***10*** of **FIG. 1**. Therefore, the host-processor ***42*** and the accelerator ***44*** (or pipeline units thereof, as discussed below) are "peers" that can transfer data vectors back and forth. Because the accelerator ***44*** does not execute program instructions, it typically performs mathematically intensive operations on data significantly faster than a bank of coprocessors can for a given clock frequency. Consequently, by combining the decision-making ability of the processor ***42*** and the number-crunching ability of the accelerator ***44***, the machine ***40*** has the same abilities as, but can often process data faster than, a conventional computing machine such as the machine ***10***. Furthermore,

as discussed below, providing the accelerator ***44*** with a communication interface that is compatible with the communication interface of the host processor ***42*** facilitates the design and modification of the machine ***40***, particularly where the processor's communication interface is an industry standard. And where the accelerator ***44*** includes one or more PLICs, the host processor ***42*** can hard configure physical interconnectors within the accelerator by sending appropriate firmware to these PLICs. The host processor ***42*** may not only configure the accelerator ***44*** in this manner during initialization of the peer-vector machine ***40***, but it may have the ability to reconfigure the accelerator during operation of the peer-vector machine as discussed below and in previously cited U.S. Patent App. Serial No. ___ entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-12-3). Moreover, the peer-vector machine ***40*** may also provide other advantages as described below and in the previously cited patent applications.

[48] Still referring to **FIG. 3**, in addition to the host processor ***42*** and the pipeline accelerator ***44***, the peer-vector computing machine ***40*** includes a processor memory ***46***, an interface memory ***48***, a pipeline bus ***50***, one or more firmware memories ***52***, an optional raw-data input port ***54***, a processed-data output port ***58***, an optional router ***61***, and a test bus ***63***.

[49] The host processor ***42*** includes a processing unit ***62*** and a message handler ***64***, and the processor memory ***46*** includes a processing-unit memory ***66*** and a handler memory ***68***, which respectively serve as both program and working memories for the processor unit and the message handler. The processor memory ***46*** also includes an accelerator-configuration registry ***70*** and a message-configuration registry ***72***, which store firmware and configuration data that respectively allow the host processor ***42*** to configure the functioning of the accelerator ***44*** and the format of the messages that the message handler ***64*** sends and receives. The configuration of the accelerator ***44*** and the message handler ***64*** is further discussed in previously cited U.S. Patent App. Serial No. ___ entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney

Docket No. 1934-12-3), and the configuration of the accelerator ***44*** is also further discussed below in conjunction with **FIGS. 4-6**.

[50] The pipeline accelerator ***44*** is disposed on at least one PLIC (**FIG. 4**) and includes hardwired pipelines $\mathbf{74_1 - 74_n}$, which process respective data without executing program instructions. The firmware memory ***52*** stores the firmware for the accelerator ***44***. More specifically, the firmware memory ***52*** stores the firmware for the PLICs that compose the accelerator ***44*** as discussed further below in conjunction with **FIGS. 4 – 6**. Alternatively, the accelerator ***44*** may be disposed on at least one ASIC, and thus may have internal interconnections that are unconfigurable once the ASIC is formed. In this alternative where the accelerator ***44*** includes no PLICs, the machine ***40*** may omit the firmware memory ***52***. Furthermore, although the accelerator ***44*** is shown including multiple pipelines $\mathbf{74_1\text{-}74_n}$, it may include only a single pipeline. In addition, although not shown, the accelerator ***44*** may include one or more processors such as a digital-signal processor (DSP). Moreover, although not shown, the accelerator ***44*** may include a data input port and/or a data output port.

[51] The general operation of the peer-vector machine ***40*** is discussed in previously cited U.S. Patent App. Serial No. ___ entitled IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-11-3), the structure and operation of the host processor ***42*** is discussed in previously cited U.S. Patent App. Serial No. _ entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-12-3), and the structure and operation of the pipeline accelerator ***44*** is discussed in previously cited U.S. Patent App. Serial Nos. _____ entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-13-3) and _ entitled PIPELINE ACCELERATOR HAVING MULTIPLE PIPELINE UNITS AND RELATED COMPUTING MACHINE AND METHOD (1934-15-3). The operating configurations of the PLICs that compose the accelerator ***44*** are discussed in previously cited U.S. Patent App. Serial No. _ entitled PIPELINE ACCELERATOR FOR

IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-13-3) and below in conjunction with **FIGS. 4 – 6**.

[52] Referring to **FIGS. 4 – 6**, techniques for "hard" configuring the accelerator ***44*** PLICs are discussed. As alluded to above, the hard configuration of a PLIC is programmed by firmware and denotes the specific physical interconnections among the components of the PLIC, *i.e.*, how one logic block is electrically connected to another logic block. This is in contrast to the "soft" configuration, which denotes a higher-level configuration of an already-hard-configured PLIC. For example, a hard-configured PLIC may include a buffer, and may also include a register that allows one to soft configure the size of the buffer by loading corresponding soft-configuration data into the register. Soft configuration of the accelerator ***44*** is further discussed in previously cited U.S. Patent App. Serial Nos. _____ entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-12-3) and __ entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-13-3).

[53] **FIG. 4** is a block diagram of a pipeline unit ***78*** of the pipeline accelerator ***44*** of **FIG. 3** according to an embodiment of the invention. The hardwired pipelines $\boldsymbol{74_1 - 74_n}$ (**FIG. 3**) are part of the pipeline unit ***78***, which, as discussed below, includes circuitry that, *e.g.,* controls the hardwired pipelines and allows them to receive, send, and store data. Although only one pipeline unit ***78*** is shown in **FIG.** 4, the accelerator ***44*** may include multiple pipeline units (each including at least some of the hardwired pipelines $\boldsymbol{74_1 - 74_n}$) as discussed in previously cited U.S. Patent App. Serial No. ________ entitled PIPELINE ACCELERATOR HAVING MULTIPLE PIPELINE UNITS AND RELATED COMPUTING MACHINE AND METHOD (Attorney Docket No. 1934-15-3). As discussed below, in one implementation, the hard configuration of the pipeline unit ***78*** is programmable with firmware. This allows one to modify the functioning of the pipeline unit ***78*** by merely modifying the firmware. Furthermore, the host processor ***42*** (**FIG. 3**) can provide the modified firmware to the pipeline unit ***78*** during an initialization or reconfiguration of the peer-vector machine ***40*** (**FIG. 3**), and

thus can eliminate the need for one to manually load the modified firmware into the pipeline unit.

[54] The pipeline unit ***78*** includes a pipeline circuit ***80***, such as a PLIC or an ASIC, the firmware memory ***52*** (where the pipeline circuit is a PLIC), and a data memory ***81***, which may all be disposed on a circuit board or card ***83***. The data memory ***81*** is further discussed in previously cited U.S. Patent App. Serial No. __ entitled PROGRAMMABLE CIRCUIT AND RELATED COMPUTING MACHINE AND METHOD (Attorney Docket No. 1934-14-3), and the combination of the pipeline circuit ***80*** and the firmware memory ***52*** forms a programmable-circuit unit.

[55] The pipeline circuit ***80*** includes a communication interface ***82***, which transfers data between a peer, such as the host processor ***42*** (**FIG. 3**), and the data memory ***81***, and also between the peer and the following other components of the pipeline circuit: the hardwired pipelines $\boldsymbol{74_1}$-$\boldsymbol{74_n}$ via a communication shell ***84***, a pipeline controller ***86***, an exception manager ***88***, and a configuration manager ***90***. The pipeline circuit ***80*** may also include an industry-standard bus interface ***91*** and a communication bus ***93***, which connects the interface ***82*** to the interface ***91***. Alternatively, the functionality of the interface ***91*** may be included within the communication interface ***82*** and the bus ***93*** omitted. The structure and operation of the hardwired pipelines $\boldsymbol{74_1}$-$\boldsymbol{74_n}$, controller ***86***, exception manager ***88***, configuration manager ***90***, and bus interface ***91*** are discussed in previously cited U.S. Patent App. Serial No. ______ entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-13-3).

[56] The communication interface ***82*** sends and receives (via the bus interface ***91*** where present) data in a format recognized by the message handler ***64*** (**FIG. 3**), and thus typically facilitates the design and modification of the peer-vector machine ***40*** (**FIG. 3**). For example, if the data format is an industry standard such as the Rapid I/O format, then one need not design a custom interface between the host processor ***42*** and the pipeline unit ***78*** . Furthermore, by allowing the pipeline unit ***78*** to communicate with other peers, such as the host processor ***42*** (**FIG. 3**), via the pipeline

bus ***50*** instead of via a non-bus interface, one can change the number of pipeline units by merely connecting or disconnecting them (or the circuit cards that hold them) to the pipeline bus instead of redesigning a non-bus interface from scratch each time a pipeline unit is added or removed.

[57] Where the pipeline circuit ***80*** is a PLIC such as an FPGA, the communication interface ***82*** includes a programming port ***94***, which allows the pipeline circuit to load firmware from the host processor ***42*** (**FIG. 3**) into the firmware memory ***52*** as discussed below. For example, if the firmware memory ***52*** is an EEPROM, then during a programming cycle the communication interface ***82*** generates, and the port ***94*** delivers, the programming signals that the firmware memory requires. Circuitry for generating such programming signals is conventional, and thus is not discussed further.

[58] The structure and operation of the communication interface ***82*** is further discussed in previously cited U.S. Patent App. Serial No. ___ entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-13-3).

[59] Still referring to **FIG. 4**, the pipeline circuit ***80*** also includes a test port ***96***, and, where the pipeline circuit is a PLIC, a hard-configuration port ***98***. The test port ***96***, which is coupled to the test bus ***63***, allows the host processor ***42*** (**FIG. 3**) to monitor the results of a self test that the pipeline circuit ***80*** may perform during initialization of the peer-vector machine ***40*** (**FIG. 3**) as discussed below. The manufacture typically includes the test port ***96*** with the pipeline circuit ***80***, and typically provides the test port with an interface (not shown) that is compatible with an industry-standard test protocol such as JTAG. The hard-configuration port ***98*** allows the pipeline circuit ***80*** to configure itself by downloading firmware from the memory ***52*** as discussed below. Like the test port ***96***, the manufacture typically includes the configuration port ***98*** with the pipeline circuit ***80***, and typically provides the configuration port with an industry-standard memory interface and state machine (neither shown) that serially downloads the firmware from a predetermined address range of the memory ***52***.

[60] As discussed above and further below, where the pipeline circuit ***80*** is a PLIC, the firmware memory ***52*** stores the firmware that represents one or more sets hard configurations of the pipeline circuit. The firmware memory ***52*** includes a test port ***104*** and programming and configuration ports ***106*** and ***108***. The test port ***104***, which is coupled to the test bus ***63***, allows the host processor ***42*** (**FIG. 3**) to monitor the results of a self test that the firmware memory ***52*** may perform during initialization of the peer-vector machine ***40*** (**FIG. 3**) as discussed below. Also as discussed below, the test port ***104*** may allow the host processor ***42*** to load firmware into the memory ***52***. The manufacture typically includes the test port ***104*** with the memory ***52***, and typically provides the test port with an interface (not shown) that is compatible with an industry-standard test protocol such as JTAG. The programming port ***106***, which is coupled to the programming port ***94*** of the communication interface ***82*** via a programming bus ***110***, allows the pipeline circuit ***80*** to load firmware into the memory ***52*** as discussed below. And the hard-configuration port ***108***, which is coupled to the hard-configuration port ***98*** of the pipeline circuit ***80*** via a configuration bus ***112***, allows the pipeline circuit to download firmware from the memory ***52*** as discussed below. Typically, the firmware memory ***52*** is a nonvolatile memory such as an EEPROM, which retains data in the absence of power. Consequently, the firmware memory ***52*** continues to store the firmware after the pipeline unit ***78*** is powered down.

[61] Still referring to **FIG. 4**, although the firmware memory ***52*** and the data memory ***81*** are described as being external to the pipeline circuit ***80***, either or both memories may be incorporated into the pipeline circuit. Where the memory ***52*** is disposed inside of the pipeline circuit ***80***, a designer may need to modify the structures of the programming and configuration busses ***110*** and ***112*** accordingly. Furthermore, although the pipeline unit ***78*** is described as having a programming bus ***110*** that is separate from the configuration bus ***112***, a single bus (not shown) may perform the functions of both the programming and configuration busses. Alternatively, the pipeline unit ***78*** may include multiple instances this single bus, or multiple instances of one or both of the programming and configuration ***112*** and ***110***.

[62] **FIG. 5** is a diagram of a logical partitioning of the firmware memory ***52*** of **FIG. 4** according to an embodiment of the invention.

[63] A section ***114*** of the memory ***52*** stores firmware that represents an initial configuration of the pipeline circuit ***80*** (**FIG. 4**). That is, when downloaded to the pipeline circuit ***80***, this firmware causes the pipeline circuit to have the initial configuration. In one implementation of the initial configuration, the pipeline circuit ***80*** includes the communication interface ***82*** (and the industry-standard bus interface ***91*** if needed) of **FIG. 4** and self-test circuitry (not shown) that executes a self test of the pipeline circuit and the data memory ***81***. The pipeline circuit ***80*** can then provide the results of the self test to the host processor ***42*** (**FIG. 3**) via the test bus ***63*** or the communication interface ***82***. The initial configuration also allows the host processor ***42*** to load modified firmware into the firmware memory ***52*** via the communication interface ***82*** and the programming bus ***110*** as discussed below.

[64] Sections $\boldsymbol{116_1 - 116_i}$ of the memory ***52*** each store firmware that represents a respective operating configuration of the pipeline circuit ***80***. Typically, the pipeline circuit ***80*** downloads the firmware from a predetermined one of the sections $\boldsymbol{116_1 - 116_i}$ at the end of the initialization of the accelerator ***44*** (**FIG. 3**). As discussed below, the pipeline circuit ***80*** may be preprogrammed to download firmware from a particular section $\boldsymbol{116_1 - 116_n}$, or the host processor ***42*** (**FIG. 3**) may instruct the pipeline circuit to download the firmware from a particular section. Typically, in each of the *i* operating configurations, the pipeline circuit ***80*** includes the components (*e.g.,* hardwired pipelines $\boldsymbol{74_1 - 74_n}$, controller ***86***) shown in **FIG. 4**. But in each of these configurations, the pipeline circuit ***80*** typically operates a differently. For example, the communication interface ***82*** may implement one protocol in one configuration and another protocol in another configuration. Or, the pipelines $\boldsymbol{74_1 - 74_n}$ may perform one set of operations on data in one configuration and perform another set of operations on the data in another configuration.

[65] Optional section ***118*** stores a description or identification of the operating configurations respectively represented by the firmware stored in the sections $\boldsymbol{116_1}$ –

116_i of the memory ***52***. This description/identification allows the host processor ***42*** (**FIG. 3**) to identify the firmware stored in the memory ***52***.

[66] Optional section ***120*** stores a profile of the pipeline unit ***78*** (**FIG. 4**). The profile typically describes the hardware layout of the pipeline unit ***78*** sufficiently for the host processor ***42*** (**FIG. 3**) to appropriately configure itself, the pipeline unit, and other peers (not shown) of the peer-vector machine ***40*** (**FIG. 3**) for intercommunication. For example, the profile may identify the data operations and communication protocols that the pipeline unit ***78*** is capable of implementing, the size of the data memory ***81***, the operating configurations represented by the firmware stored in sections ***116_1 – 116_i*** (if the section ***118*** is omitted), and a currently desired operating configuration. Consequently, by reading the profile during initialization of the peer-vector machine ***40***, the host processor ***42*** can properly configure the message handler ***64*** (**FIG. 3**) to communicate with the pipeline unit ***78***. Furthermore, the host processor ***42*** may select the section ***116_1 – 116_i*** of firmware that the pipeline circuit ***80*** should download. Or, if none of this firmware is suitable, the host processor ***42*** may load modified firmware into the memory ***52***. This technique is analogous to the "plug and play" technique by which a computer can configure itself to communicate with a newly installed peripheral such as a disk drive.

[67] Alternatively, the section ***120*** may store a profile identifier — often called a "running index" — that allows the host processor ***42*** (**FIG. 3**) to retrieve the profile from a table that is stored in, *e.g.*, the accelerator configuration registry ***70*** (**FIG. 3**). The running index is typically a number, much like a model number of a product, which the host processor ***42*** can match to a stored profile.

[68] In yet another alternative, the pipeline unit ***78*** (**FIG. 4**) may store the profile identifier in a "hardwired" form to eliminate the chance that one may inadvertently overwrite the profile in the section ***120***. For example, the pipeline unit ***78*** may store the profile identifier in a hardwired "register" that the host processor ***42*** (**FIG. 3**) can read via the test bus ***63***, or via the pipeline bus ***50*** and the pipeline circuit ***80*** (**FIG. 4**). This

register may be formed from, *e.g.*, electro-mechanical switches, jumpers, or soldered connections (not shown).

[69] Still referring to **FIG. 5**, optional section ***122*** of the firmware memory ***52*** may store miscellaneous data, such as a self-test routine that the firmware memory ***52*** runs during initialization of the accelerator ***44***.

[70] Referring to **FIGS. 3 – 5**, the operation of the peer-vector machine ***40*** — particularly the operation of the host processor ***42***, pipeline circuit ***80***, and firmware memory ***52*** — is discussed below according to an embodiment of the invention.

[71] When the peer-vector machine ***40*** is first powered on, the host processor ***42*** initializes itself as discussed in previously cited U.S. Patent App. Serial No. _______ entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-12-3), and the accelerator ***44*** partially initializes itself. More specifically, during this partial initialization, the pipeline circuit ***80*** downloads the initial-configuration firmware from the section ***114*** of the memory ***52***. As discussed above, in the initial configuration, the pipeline circuit ***80*** includes at least the communication interface ***82*** and test circuitry (not shown). After the pipeline circuit ***80*** is configured in the initial configuration, the test circuitry performs a self test of the pipeline circuit and the data memory ***81***, and provides the results of the self test to the host processor ***42*** via the test port ***96*** and the test bus ***63***. The firmware memory ***52*** may also perform a self test and provide the results to the host processor ***42*** via the test port ***104*** and the test bus ***63*** as discussed above in conjunction with **FIG. 5**.

[72] Next, the host processor ***42*** determines if an exception occurred during the partial initialization of the accelerator ***44***. For example, the host processor ***42*** analyzes the self-test results from the test bus ***63*** to determine whether the pipeline circuit ***80***, the data memory ***81***, and the firmware memory ***52*** are functioning properly.

[73] If an exception did occur, then the host processor ***42*** handles it in a predetermined manner. For example, if the host processor ***42*** does not receive a self-test result from the pipeline circuit ***80***, then it may check, via the test bus ***63***,

whether the initial-configuration firmware is stored in the section ***114*** of the firmware memory ***52***. If the initial-configuration firmware is not stored, then the host processor ***42*** may load the initial-configuration firmware into the section ***114*** via the pipeline bus ***50*** or the test bus ***63***, cause the pipeline circuit ***80*** to download this firmware, and then analyze the result of the self test. The host processor's handling of exceptions is further discussed in previously cited U.S. Patent App. Serial No. ___ entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-12-3).

[74] If no exception occurred, then host processor ***42*** reads the profile identifier from the pipeline unit ***78***, and subsequently obtains the corresponding profile of the pipeline unit from the accelerator configuration registry ***70***. Obtaining the profile from the registry ***70*** instead of from the section ***120*** of the firmware memory ***52*** is often preferred, because if the pipeline circuit ***80*** is an ASIC, then the pipeline unit ***78*** may not include a nonvolatile memory such as the firmware memory. If the profile identifier indicates that the pipeline circuit ***80*** is an ASIC, then the host processor ***42*** determines that no firmware need be downloaded to the pipeline circuit. Alternatively, the host processor ***42*** (**FIG. 3**) may obtain the profile from the section ***120*** of the firmware memory ***52***. In this alternative, it is unnecessary for the pipeline unit ***78*** to store a profile identifier, although the pipeline unit may store a profile identifier in case the profile is inadvertently deleted from the section ***120***.

[75] Next, after reading the profile identifiers from all of the pipeline units ***78*** (only one shown in **FIG. 4**), the host processor ***42*** effectively generates a map of all the pipeline units ***78*** in the accelerator ***44***, and stores this map, *e.g.*, in the handler memory ***68***.

[76] Then, for each pipeline unit ***78***, the host processor ***42*** extracts from the profile the identity of the desired operating configuration of the pipeline circuit ***80***. Extracting the desired operating configuration during initialization of the accelerator ***44*** allows one to modify the operation of the pipeline circuit ***80*** by merely updating the profile prior to the initialization.

[77] Next, the host processor ***42*** determines whether the firmware that represents the desired operating configuration is stored in the firmware memory ***52***. For example, the host processor ***42*** can read the configuration description from the memory section ***118*** via the programming bus ***110*** and the communication interface ***82*** — because the pipeline circuit ***80*** is in the initial configuration, the communication interface is present — to determine whether the desired firmware is stored in any of the sections $\mathbf{116_1} - \mathbf{116_i}$. Alternatively, the host processor ***42*** may read the configuration description directly from the memory ***52*** via the test bus ***63*** and the test port ***104***.

[78] If the firmware that represents the desired operating configuration is not stored in the firmware memory ***52***, then the host processor ***42*** loads this firmware from the accelerator configuration registry ***70*** into one of the sections $\mathbf{116_1} - \mathbf{116_i}$ of the firmware memory via the communication interface ***82***, the programming ports ***94*** and ***106***, and the programming bus ***110***. If the firmware is not in the registry ***70***, then the host processor ***42*** may retrieve the firmware from an external library (not shown), or may generate an exception indicator so that a system operation (not shown) can load the firmware into the registry ***70***.

[79] Next, the host processor ***42*** instructs the pipeline circuit ***80*** to download the desired firmware from the corresponding section $\mathbf{116_1} - \mathbf{116_i}$ of the memory ***52*** via the port ***108***, the configuration bus ***112***, and the port ***98***.

[80] After the pipeline circuit ***80*** downloads the desired firmware, it is in the desired operating configuration and is ready to begin processing data. But even after the pipeline circuit ***80*** is in its desired operating configuration, the host processor ***42*** may load new firmware into the sections $\mathbf{116_1} - \mathbf{116_i}$ of the memory ***52*** via the communication interface ***82*** or via the test bus ***63***. For example, to load new firmware, the host processor ***42*** may first cause the pipeline circuit ***80*** to reload the firmware from the section ***114*** of the memory ***52*** so that the pipeline circuit is again in the initial configuration. Then, the host processor ***42*** loads the new firmware into one of the sections $\mathbf{116_1} - \mathbf{116_i}$ via the pipeline bus ***50*** and the communication interface ***82***. Next, the host processor ***42*** causes the pipeline circuit ***80*** to download the new firmware so

that the pipeline circuit is in the new operating configuration. Allowing the pipeline circuit ***80*** to load new firmware into the memory ***52*** only when in the initial configuration provides two advantages. First, it prevents the pipeline circuit ***80*** from inadvertently altering the firmware stored in the memory ***52*** when the pipeline circuit is in an operating configuration. Second, it allows the operating configurations to utilize resources of the pipeline circuit ***80*** that would otherwise be used for the circuitry needed to write firmware to the memory ***52***.

[81] **Fig. 6** is a block diagram of a pipeline unit ***124*** of the pipeline accelerator ***44*** of **FIG. 3** according to another embodiment of the invention.

[82] The pipeline unit ***124*** is similar to the pipeline unit ***78*** of **FIG. 4** except that the pipeline unit ***124*** includes multiple pipeline circuits ***80*** — here two pipeline circuits ***80a*** and ***80b*** — and multiple firmware memories — here two memories ***52a*** and ***52b***, one memory for each pipeline circuit. The combination of the pipeline circuits ***80a*** and ***80b*** and the firmware memories ***52a*** and ***52b*** forms a programmable-circuit unit. In one implementation, each of the memories ***52a*** and ***52b*** is partitioned as shown in **FIG. 5**, except that the firmware memory ***52b*** may omit the section ***120***, which stores the profile of the pipeline unit ***124*** and which otherwise would be redundant with the section ***120*** of the memory ***52a***. Alternatively, the pipeline circuits ***80a*** and ***80b*** may share a single firmware memory that includes respective sections that are operatively similar to the memories ***52a*** and ***52b***. Increasing the number of pipeline circuits ***80*** typically allows an increase in the number *n* of hardwired pipelines 74_1-74_n, and thus an increase in the functionality of the pipeline unit ***124*** as compared to the pipeline unit ***78***. Furthermore, either one or both of the pipeline circuits ***80a*** and ***80b*** may be an ASIC, in which case the corresponding firmware memory(ies) ***52*** may be omitted.

[83] Further details of the structure and operation of the pipeline unit ***124*** are discussed in previously cited U.S. Patent App. Serial No. ___ entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-13-3).

[84] The pipeline circuit ***80a*** includes a test port ***96a*** and a hard-configuration port ***98a***, which are respectively similar to the test port ***96*** and hard-configuration port ***98*** of **FIG. 4**. And like the pipeline circuit ***80*** of **FIG. 4**, the pipeline circuit ***80a*** includes the communication interface ***82*** having the programming port ***94***.

[85] The pipeline circuit ***80b*** includes a test port ***96b*** and a hard-configuration port ***98b***, which are also respectively similar to the test port ***96*** and the hard-configuration port ***98*** of **FIG. 4**. And because the host processor ***42*** (**FIG. 3**) can program the firmware memory ***52b*** via the communication interface ***82*** of the pipeline circuit ***80a*** as discussed below, the pipeline circuit ***80b*** does not include a programming port.

[86] The firmware memory ***52a*** includes test, programming, and hard-configuration ports ***104a***, ***106a***, and ***108a***, which are respectively similar to the test, programming, and hard-configuration ports ***104***, ***106***, and ***108*** of **FIG. 4**. The test port ***104a*** is coupled to the test bus ***63***, the programming port ***106a*** is coupled to the programming port ***94a*** of the communication interface ***82*** via the programming bus ***110***, and the hard-configuration port ***108a*** is coupled to the hard-configuration port ***98a*** of the pipeline circuit ***80a*** via a configuration bus ***112a***.

[87] Likewise, the firmware memory ***52b*** includes test, programming, and hard-configuration ports ***104b***, ***106b***, and ***108b***, which are respectively similar to the test, programming, and hard-configuration ports ***104***, ***106***, and ***108*** of **FIG. 4**. The test port ***104b*** is coupled to the test bus ***63***, the programming port ***106b*** is coupled to the programming port ***94a*** of the communication interface ***82*** via the programming bus ***110***, and the hard-configuration port ***108b*** is coupled to the hard-configuration port ***98b*** of the pipeline circuit ***80b*** via a configuration bus ***112b***.

[88] Referring to **FIGS. 3, 5** and **6**, the operation of the peer-vector machine ***40*** — particularly the host processor ***42***, the pipeline circuits ***80a*** and ***80b***, and the firmware memories ***52a*** and ***52b*** — is discussed below according to an embodiment of the invention.

[89] When the peer-vector machine ***40*** is first powered on, the host processor ***42*** initializes itself as discussed in previously cited U.S. Patent App. Serial No. ________ entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-12-3), and the accelerator ***44*** partially initializes itself. More specifically, during this partial initialization, the pipeline circuits ***80a*** and ***80b*** download initial-configuration firmware from the sections ***114a*** and ***114b*** of the firmware memories ***52a*** and ***52b***, respectively. In the respective initial configurations, the pipeline circuit ***80a*** includes at least the communication interface ***82*** and test circuitry (not shown), and the pipeline circuit ***80b*** includes at least test circuitry (not shown). After the pipeline circuits ***80a*** and ***80b*** are configured in their respective initial configurations, the test circuit within each pipeline circuit performs a respective self test of the pipeline circuit — the test circuitry of one or both of the pipeline circuits ***80a*** and ***80b*** may also test the data memory ***81*** — and provides the results of these self tests to the host processor ***42*** via the test ports ***96a*** and ***96b***, respectively, and the test bus ***63***. The firmware memories ***52a*** and ***52b*** may also perform respective self tests and provide the results to the host processor ***42*** via the test ports ***104a*** and ***104b***, respectively, and the test bus ***63*** as discussed above in conjunction with **FIG. 5**.

[90] Next, the host processor ***42*** determines if an exception occurred during the partial initialization of the accelerator ***44***. For example, the host processor ***42*** analyzes the self-test results from the test bus ***63*** to determine whether the pipeline circuits ***80a*** and ***80b***, the data memory ***81***, and the firmware memories ***52a*** and ***52b*** are functioning properly.

[91] If an exception did occur, then the host processor ***42*** handles it in a predetermined manner. For example, if the host processor ***42*** does not receive a self-test result from the pipeline circuit ***80a***, then it may check, via the test bus ***63***, whether the initial-configuration firmware is stored in the section ***114a*** of the firmware memory ***52a***. If the initial-configuration firmware is not stored, then the host processor ***42*** may load the initial-configuration firmware into the section ***114a***, cause the pipeline circuit ***80a*** to download this firmware, and then analyze the result of the self

test. This example also applies to the pipeline circuit ***50b*** and the firmware memory ***52b***. The host processor's handling of exceptions is further discussed in previously cited U.S. Patent App. Serial No. __ entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-12-3).

[92] If no exception occurred, then the host processor ***42*** reads the profile identifier from the pipeline unit ***124***, and subsequently obtains the corresponding profile of the pipeline unit from the accelerator configuration registry ***70*** or from the section ***120*** of the firmware memory ***52a*** as discussed above in conjunction with **FIG. 4**.

[93] Next, after reading the profile identifiers from all of the pipeline units ***124*** (only one shown in **FIG. 4**), the host processor ***42*** effectively generates a map of all the pipeline units in the accelerator ***44***, and stores this map, *e.g.*, in the handler memory ***68***.

[94] Then, the host processor ***42*** extracts from the profile the identities of the desired operating configurations of the pipeline circuits ***80a*** and ***80b***. Extracting the desired operating configurations during initialization of the accelerator ***44*** allows one to modify the operation of the pipeline circuit ***80a*** and/or ***80b*** by merely updating the profile prior to the initialization.

[95] Next, the host processor ***42*** determines whether the firmware that represents the desired operating configurations is stored in the firmware memories ***52a*** and ***52b***. For example, the host processor ***42*** can read the configuration description from the memory section ***118a*** of the memory ***52a*** via the programming bus ***110*** and the communication interface ***82*** — because the pipeline circuit ***80a*** is in the initial configuration, the communication interface is present — to determine whether the desired firmware is stored in any of the sections $\boldsymbol{116a_1} - \boldsymbol{116a_i}$. Alternatively, the host processor ***42*** may read the configuration description directly from the memory ***52a*** via the test bus ***63*** and the test port ***104a***. This example also applies to the pipeline circuit ***50b*** and the firmware memory ***52b***.

[96] If the firmware that represents one or both of the desired operating configurations is not stored in the firmware memories ***52a*** and/or ***52b***, then the host

processor ***42*** loads this firmware from the accelerator configuration registry ***70*** into one of the sections ***116***$_1$ – ***116***$_i$ of the appropriate firmware memory via the communication interface ***82***, the programming ports ***94*** and ***106***, and the programming bus ***110***. For example, if the firmware that represents the desired operating configuration of the pipeline circuit ***80b*** is not stored in the memory ***52b***, then the host processor ***42*** loads this firmware from the registry ***70*** into one of the sections ***116b***$_1$ – ***116b***$_i$ via the interface ***82***, programming ports ***94*** and ***106b***, and the programming bus ***110***. If the firmware is not in the registry ***70***, then the host processor ***42*** may retrieve the firmware from an external library (not shown), or may generate an exception indicator so that a system operator (not shown) can load the firmware into the registry ***70***.

[97] Next, the host processor ***42*** instructs the pipeline circuit ***80a*** to download the desired firmware from the corresponding sections ***116a***$_1$- ***116a***$_i$ of the memory ***52a*** via the port ***108a***, the configuration bus ***112a***, and the port ***98a***, and instructs the pipeline circuit ***80b*** to download the desired firmware from the corresponding sections ***116b***$_1$ - ***116b***$_i$ of the memory ***52b*** via the port ***108b***, the configuration bus ***112b***, and the port ***98b***.

[98] After the pipeline circuits ***80a*** and ***80b*** download the desired firmware, they are in the desired operating configurations, and are ready to begin processing data. But even after the pipeline circuits ***80a*** and ***80b*** are in their desired operating configurations, the host processor ***42*** may load new firmware into the sections ***116***$_1$ – ***116***$_i$ of the memories ***52a*** and ***52b*** via the communication interface ***82*** or via the test bus ***63*** in a manner similar to that discussed above in conjunction with **FIG. 4**.

[99] The preceding discussion is presented to enable a person skilled in the art to make and use the invention. Various modifications to the embodiments will be readily apparent to those skilled in the art, and the generic principles herein may be applied to other embodiments and applications without departing from the spirit and scope of the present invention. Thus, the present invention is not intended to be limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.